Given a bipartite graph $G = (V, W, E)$ s.t. $|V| = |W| = n$, the **Perfect Match (PM)** problem is a decision problem $L$ s.t. a bipartite graph $G \in L \iff G$ has a perfect matching; i.e. a subset $E' \subseteq E$ s.t. every $x \in V$ touches exactly one edge $e \in E'$.

We present the notions of parallel and concurrent computation, and construct a randomized algorithm with poly-logarithmic runtime for both the decision problem (does $G$ have a perfect matching?) and the construction problem (construct a perfect matching or return $null$ if no match exists).

## 1 The CREW Model

**Definition 1** Given a language $L$ (some decision problem) and some input $x \in \{0, 1\}^n$, the **PRAM** model assumes we have a polynomial $p(n)$ number of CPUs and a polynomial $q(n)$ sized shared memory. In every 'clock cycle', each CPU may (in parallel):

1. Read a cell of the shared memory
2. Perform some (short, constant time) local computation
3. Write to a cell in the shared memory

**Definition 2** The **CREW** model stands for Concurrent Reads Exclusive Writes, which simplifies the **PRAM** model somewhat by exempting us from discussing race conditions.

**Example 3** Matrix product.

For some $A, B \in \mathbb{R}^{n \times n}$ we can take $n^2$ CPU cores; core $(i,j)$ will compute:

$$(AB)_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$$

As these all run in parallel and each core computes $n$ sums and products, with $n^2$ cores we get a parallel runtime of $O(n)$. This is still polynomial runtime; a naive non-parallel approach also runs in polynomial ($O(n^3)$) time. Can we do better?

Fix some $i, j$. Core $(i, j)$ must compute $\sum_k A_{ik}B_{kj}$; if we use $n$ cores to compute the $n$ products in parallel and then perform the addition in a tree-like way, we can improve the runtime to $O(\log(n))$. To illustrate, assume $n = 2^r$ for some $r \in \mathbb{N}$.
The depth of the above computation is \( r = O(\log(n)) \). By adding \( n \) cores per previous core, we now have a parallel logarithmic-time algorithm utilizing \( O(n^3) \) cores; much better than linear time.

### 1.1 Remarks:

- The above product/sum tree can be described as a circuit; the complete circuit that computes the matrix product is of size \( O(n^3) \) and depth \( O(\log(n)) \)

## 2 Decision problem

The next algorithm is from [2].

### 2.1 Algorithm

\( PM \) is the decision problem that given \( G = (V, E) \) accepts \( \iff \) \( G \) has a perfect matching. We will discuss only those cases where \( G \) is a bipartite graph; we shall henceforth denote \( G = (V, W, E) \) where \( |V| = |W| = n \) are the two sides (node sets) of the graph.

**Remark 4** The problem of counting how many such perfect matchings exist is \( \#P \)-complete!

Our algorithm begins with the definition of the following matrix of variables:

\[
M_{ij} = \begin{cases} 
0 & (i, j) \not\in E \\
x_{ij} & (i, j) \in E 
\end{cases}
\]

where \( x_{ij} \) denote \( n^2 \) distinct variables. The algorithm does as follows:

1. For every \( i, j \) fix some random \( a_{ij} \in \{1, 2, 3, \ldots, 2n\} \)
2. Substitute every \( x_{ij} \) in \( M \) as \( a_{ij} \) (zero entries remain zero) to create the matrix \( A \):

\[
A_{ij} = \begin{cases} 
0 & (i, j) \not\in E \\
a_{ij} & (i, j) \in E 
\end{cases}
\]
3. Compute \( \det(A) \)
4. Accept $\iff \det(A) \neq 0$

**Definition 5** Given a matrix $A \in \mathbb{R}^{n \times n}$, the determinant $\det(A)$ is defined:

$$\det(A) = \sum_{\pi \in S_n} \text{sign}(\pi) \prod_{i=1}^{n} A_{i,\pi(i)}$$

where $S_n$ denotes the set of all permutations on the set $\{1, \ldots, n\}$.

**Proposition 6** If there is no perfect matching, the above algorithm always rejects.

**Proof:** Assume there is no perfect matching and let $\pi \in S_n$. If, for every $i$, we have $(i, \pi(i)) \in E$, then the set $\{(i, \pi(i)) : 1 \leq i \leq n\}$ is a perfect matching; hence there must exist some $i$ s.t. $(i, \pi(i)) \notin E$ and therefore $M_{i,\pi(i)} = 0$ by definition. As such, for any $\vec{a} = (a_{11}, \ldots, a_{nn}) \in \{1, 2, \ldots, 2n\}^n$ chosen at step 1 of the algorithm:

$$\prod_{j=1}^{n} A_{j,\pi(j)} = 0 \cdot \prod_{j \neq i} A_{j,\pi(j)} = 0$$

This is true for any $\pi \in S_n$, so for any $\vec{a}$ chosen:

$$\det(A) = \sum_{\pi \in S_n} \text{sign}(\pi) \prod_{i=1}^{n} A_{i,\pi(i)} = \sum_{\pi \in S_n} \text{sign}(\pi) \cdot 0 = 0$$

and the algorithm will reject. $\blacksquare$

**Proposition 7** If a perfect matching exists, the algorithm accepts with probability $\frac{1}{2}$ or better.

To prove this proposition we recall the following lemma:

**Lemma 8 (Schwartz-Zippel lemma)** Given a field $F$, a subset $A \subseteq F$ and a polynomial $0 \neq p \in F[x_1, \ldots, x_m]$:

$$\Pr_{(a_1, \ldots, a_m) \in A^m}(p(a_1, \ldots, a_m) = 0) \leq \frac{\deg(p)}{|A|}$$

Note that it is important that $p \neq 0$!

**Proof (Prop. 7):** If a perfect matching $\{(i_k, j_k) : 1 \leq k \leq n\}$ exists, it can be defined by some permutation $\pi(i_k) = j_k$. Fix such a permutation $\pi$.

Observe $\det(M)$. As $M$ is a matrix of variables, $\det(M)$ is a polynomial in $n^2$ variables:

$$\det(M) = p(x_{11}, \ldots, x_{nn})$$

By the definition of the determinant as stated above, one of the summed elements is:

$$\text{sign}(\pi) \prod_{i=1}^{n} x_{i,\pi(i)}$$

This term is not zero (as $\pi$ defines a perfect matching) and cannot be canceled out by any other term of $p$ (because different permutations define different variables appearing in the term), so we know $p \neq 0$. We also know that each non-zero term is a product of $n$ variables, so $\deg(p) = n$. By applying SZ-lemma:

$$\Pr_{(a_{11}, \ldots, a_{nn}) \in \{1, 2, \ldots, 2n\}^{n^2}}(p(a_{11}, \ldots, a_{nn}) = 0) \leq \frac{\deg(p)}{\{|1, 2, \ldots, 2n|\}} = \frac{n}{2n} = \frac{1}{2}$$

3
This is precisely the probability of rejecting! Step 1 of the algorithm randomly chooses a vector \( \vec{a} \in \{1, ..., 2n\}^n \) and rejects \( \iff \det(A) = p(\vec{a}) = 0. \]

It remains to be shown that we may compute \( \det(A) \) efficiently. Note that up to this point we did not introduce parallelism (maybe only to generate the \( O(n^2) \) random values of \( A \) in logarithmic time); parallelism will be employed heavily to compute the determinant.

Although we will not prove this method formally, a general outline and ideas employed will be presented.

### 2.2 Computing \( \det(A) \) in parallel

We give highlights of Csanky’s algorithm [1]. Denote \( \lambda_1, ..., \lambda_n \) as the eigenvalues of \( A \). We know \( \det(A) = \prod_i \lambda_i \), so \( \det(A) \) is symmetric in \( \lambda_1, ..., \lambda_n \).

The space of symmetric functions on \( \lambda_i \) can be expressed using the following basis of symmetric functions:

\[
\begin{align*}
\sum_{i=1}^n \lambda_i & \\
\sum_{i<j} \lambda_i \lambda_j & \\
\sum_{i<j<k} \lambda_i \lambda_j \lambda_k & \\
& \quad \quad \vdots \\
\prod_{i=1}^n \lambda_i &
\end{align*}
\]

However, there exists another basis for all symmetric functions:

\[
\begin{align*}
\sum_{i=1}^n \lambda_i & = Tr(A) \\
\sum_{i=1}^n \lambda_i^2 & = Tr(A^2) \\
\sum_{i=1}^n \lambda_i^3 & = Tr(A^3) \\
& \quad \quad \vdots \\
\sum_{i=1}^n \lambda_i^n & = Tr(A^n)
\end{align*}
\]

This can be parallelized easily by computing the matrices \( A^i \) and their trace values (poly-logarithmic time, using the example above with matrix products); using this basis to compute \( \prod_i \lambda_i = \det(A) \) can also be parallelized and \( \det(A) \) can be computed in parallel, logarithmic time.

**Remarks:**

- It is yet unknown if there exists a *deterministic*, parallel polylog-time algorithm for the PM problem
- Why do we not simply compute \( p(\vec{x}) = \det(M) \) and test if \( p \equiv 0? \)
  
  \( p \) has \( n! \) terms; testing if \( p \equiv 0 \) implies we know all \( n! \) terms are zero. Merely describing such a polynomial in memory would require \( \Omega(n!) \) memory!
Also, if we could describe $p$ fully, we could count the number of non-zero terms and the result would be the number of perfect matchings on the graph - computation of which is a $\#P$-complete problem as remarked in the previous section.

3 Construction problem

Given a bipartite graph $G = (V, W, E)$, we would like to construct a perfect matching (if such a matching exists). The following sequential approach could be applied:

1. Set $X \leftarrow \phi$
2. Check if $G$ has a perfect matching. If not, return "no PM"
3. Choose some $e \in E$ and define $G' = (V, W \setminus \{e\})$.
4. Check if $G'$ has a perfect matching:
   • If so, set $G \leftarrow G'$ and return to step 3.
   • Otherwise:
     (a) Set $X \leftarrow X \cup \{e\}$
     (b) Remove $e$ and both the vertices it touches from $G$
     (c) Return to step 3

This works, but requires polynomial runtime and is sequential in execution (no obvious way to parallelize).

However, if we know that there is precisely one perfect matching $X$, we can improve on this algorithm by testing all edges $e \in E$ in parallel: if there is no perfect match without $e$ then it must be $e \in X$, and if there is a perfect match without $e$ then by uniqueness of $X$ it follows that $e \notin X$.

Applying the efficient parallel algorithm for the decision problem in $|E|$ separate instances, we can efficiently construct a perfect matching.

We must now reduce the problem to the case where there is at most one perfect matching.

3.1 The Isolation Lemma

The Isolation Lemma is from [2].

**Lemma 9 (The Isolation Lemma)** Let $X$ be some set of $n$ elements and fix some $\Omega \subseteq 2^X$ (where $2^X = \{S : S \subseteq X\}$). Let $m \in \mathbb{N}$. For a given weight function $w : X \to [m]$ (where $[m] = \{1, 2, \ldots, m\}$) define the natural extension $w : 2^X \to \{1, \ldots, m\}$ as $w(S) = \sum_{x \in S} w(x)$. We say the minimal set in $\Omega$ is uniquely defined if there exists a unique set $S \in \Omega$ s.t. $w(S) = \min\{w(S') : \emptyset \neq S' \in \Omega\}$.

When $w$ is sampled uniformly over all possible weight functions:

$$\Pr_{w : X \to [m]}(\text{the minimal set in } \Omega \text{ is uniquely defined}) \geq (1 - \frac{1}{m})^n$$

**Example 10** $\Omega = 2^X$.

In this case there are $2^n - 1$ sets to consider (disregarding the empty set, the definition of uniquely defined minimum does not consider the empty set) but values of any $w$ range in $0 < w(S) \leq n \cdot m$; many more sets than possible weight values. However, as $w(x) \geq 1$ for any $x \in X$, if $S_1 \subseteq S_2$ then $w(S_1) < w(S_2)$. When $\Omega = 2^X$ this means we should only consider singletons (sets of cardinality 1) and the isolation lemma becomes intuitive (and easy to prove).
Example 11 $\Omega = \{ S \subseteq X : |S| = \frac{1}{2}n \}$ (n even).

We get $|\Omega| = \left(\binom{n}{\frac{1}{2}n}\right) \approx 2^n \sqrt{n}$, which is still a very large number of sets. Intuitively, we might expect that each possible value is obtained many times. To see why we should be more cautious, consider what happens if we toss $2^n$ independent coins, each taking a random value in $\{1, \ldots, m\}$. Then, we expect the result to be heavily centered around the mean $\frac{1}{2}m$. The questions whether we get a unique minimal (or maximal) values is then delicate. However, in our case the $2^n$ values are dependent (only the weights to the base elements are independent). The isolation lemma tells us we get a unique minimum/maximum with a good probability.

Example 12 Given a bipartite graph $G = (V, W, E)$ where $|V| = |W| = n$, denote $X = E$ and set:

$$\Omega = \{ \hat{E} \subseteq E : \hat{E} \text{ is a perfect matching} \}$$

By definition, every $S \in \Omega$ satisfies $|S| = n$. By applying the isolation lemma, if we randomly set values $w(e) \in [2n]$ for every $e \in E$, with constant probability there exists a perfect matching in $\Omega$ that is minimal and unique.

Proof (The Isolation Lemma): Denote $W = \{ w : X \to [m] \}$ (the set of all weight functions) and define a 'good' subset of $W$:

$$\text{Good} = \{ w \in W : \text{the minimal set is uniquely obtained} \}$$

We call a weight function $w$ non-borderline if $w(x) \geq 2$ for every $x \in X$ and denote by $C$ the set of all non-borderline weight functions.

It’s easy to see that $|C| = (m-1)^n$, and the density of $C$ in $W$ (which we denote $\rho_W(C)$) is:

$$\rho_W(C) = \frac{|C|}{|W|} = \frac{(m-1)^n}{m^n} = (1 - \frac{1}{m})^n$$

We shall now prove $|\text{Good}| \geq |C|$ by defining a 1-1 mapping $\phi : C \to \text{Good}$. Given some $w \in C$ set some $S \in \text{argmin}_{S \in \Omega}(w(S'))$ ($S$ is not necessarily unique), define:

$$\hat{w}(x) = \begin{cases}  
  w(x) - 1 & x \in S \\
  w(x) & x \notin S
\end{cases}$$

Note that $\hat{w} : X \to [m]$ is well defined because $w \in C$ and therefore $w(x) \geq 2$. It’s easy to see that $S$ is unique and minimal in $\Omega$ relative to $\hat{w}$: $S$ is the only set that has had $|S|$ subtracted from it’s previous value respective to $w$, other than strict supersets of $S$ which must have strictly larger weights. By choosing a minimal set $S$ for every $w \in C$, the mapping $\phi(w) = \hat{w}$ is a well defined mapping $\phi : C \to \text{Good}$.

Given some $\hat{w} \in \text{Good}$, the minimal set $S \in \Omega$ is unique respective to $\hat{w}$; by defining:

$$w(x) = \begin{cases}  
  \hat{w}(x) + 1 & x \in S \\
  \hat{w}(x) & x \notin S
\end{cases}$$

we get a uniquely defined source $w$ s.t. $\phi(w) = \hat{w}$, proving $\phi$ is 1-1 and therefore $|\text{Good}| \geq |C|$. Finally:

$$\Pr_{w \in W}(\text{Minimal } S \in \Omega \text{ is unique}) = \rho_W(\text{Good}) = \frac{|\text{Good}|}{|W|} \geq \frac{|C|}{|W|} = (1 - \frac{1}{m})^n$$
3.2 Algorithm

How do we find a perfect matching?

Given the matrix of variables $M$ from section 2.1, replace each $x_{ij}$ by $2^{w_{ij}}$ after randomly sampling $n^2$ values $w_{ij} \in [2n]$. We’ll get:

$$\det(M) = \sum_{\pi \in S_n} \text{sign}(\pi) \prod_{i=1}^{n} M_{i,\pi(i)}$$

Recall that if $\pi \not\in PM$ (the permutation $\pi$ doesn’t define a perfect matching $(i, \pi(i))$) then $\prod M_{i,\pi(i)} = 0$; otherwise we get $\prod M_{i,\pi(i)} = \prod x_{i,\pi(i)}$:

$$= \sum_{\pi \in PM} \text{sign}(\pi) \prod_{i=1}^{n} x_{i,\pi(i)}$$

$$= \sum_{\pi \in PM} \text{sign}(\pi) \prod_{i=1}^{n} 2^{w_{i,\pi(i)}}$$

$$= \sum_{\pi \in PM} \text{sign}(\pi)2^{w(\pi)}$$

where $w(\pi)$ is the weight of the perfect matching defined by $\pi$. The isolation lemma tells us that with a good probability the minimal weight matching is unique.

Also note that if we observe the binary representation of $\det(M)$, assuming there is a minimal unique perfect matching $\pi$ then $2^{w(\pi)}$ is the smallest value in the sum and it appears exactly once, so $w(\pi)$ is the index of the least significant set bit in the binary representation of $\det(M)$.

We arrive at the following algorithm:

1. Set $S \leftarrow \phi$

2. Choose (in parallel) random values $w_{ij} \in [2n]$

3. Replace $x_{ij} \leftarrow 2^{w_{ij}}$ in $M$ and calculate $\det(M)$

4. Check the least significant set bit $i$ in the binary representation of $\det(M)$

5. In parallel, for each $e \in E$ remove $e$ from the graph, construct the new $M'$ and repeat steps 3 and 4 (do not change $w$) to get $i'$:

   - If $i' \neq i$, add $e$ to $S$

6. Output $S$

As mentioned before the algorithm, at step 4 we know $i$ is equal to the weight of the unique minimal perfect matching (assuming it’s unique).

If we remove some $e \in \pi$ (where $\pi$ is the minimal perfect matching) from $G$, the new minimal value must be greater than $i$ because $\pi$ was unique in reaching $i$; so if $i' \neq i$ then this implies $e \in \pi$. This proves $S \subseteq \pi$.

If we remove some $e \not\in \pi$ from $G$, then $i' = i$ because no new matchings were created so $\pi \subseteq S$, and the returned set $S$ is a perfect matching!

If the minimal set is not unique, the resulting $S$ may not be a perfect matching (this can be tested during runtime in parallel $O(1)$ time). However, as the probability of sampling a weight function that induces a unique minimal set is constant, by running the entire algorithm in a polynomial number of separate instances and simply choosing any PM one of them outputs, we can greatly reduce the probability of outputting "no
PM" when a PM exists (reduce probability of false negatives to almost zero).

Steps 1-4 can be done in parallel, logarithmic time, as shown previously. Step 5 is one more parallel iteration of steps 3-4, yielding a logarithmic runtime for construction of a perfect matching!

Remarks:

- Step 5 of this algorithm improves upon the sequential approach presented in section 1.3 by parallelizing all checks for each $e \in E$. This became possible once identified a unique solution to search for - the original problem has no ‘minimality’ requirement.

References
